CS 189/289A Introduction to Machine Learning
Spring 2022 Jonathan Shewchuk Final

- Please do not open the exam before you are instructed to do so.
- **Electronic devices are forbidden on your person**, including phones, laptops, tablet computers, headphones, and calculators. Turn your cell phone off and **leave all electronics at the front of the room**, or **risk getting a zero** on the exam.
- When you start, the **first thing you should do** is **check that you have all 11 pages and all 6 questions**. The second thing is to please **write your initials at the top right of every page after this one** (e.g., write "JS" if you are Jonathan Shewchuk).
- The exam is closed book, closed notes except your two cheat sheets.
- You have **180 minutes**. (If you are in the DSP program and have an allowance of 150% or 200% time, that comes to 270 minutes or 360 minutes, respectively.)
- Mark your answers on the exam itself in the space provided. Do **not** attach any extra sheets. If you run out of space for an answer, write a note that your answer is continued on the back of the page.
- The total number of points is 150. There are 15 multiple choice questions worth 4 points each, and 5 written questions worth a total of 90 points.
- For multiple answer questions, fill in the bubbles for **ALL correct choices:** there may be more than one correct choice, but there is always at least one correct choice. **NO partial credit** on multiple answer questions: the set of all correct answers must be checked.

First name	
Last name	
SID	
First and last name of student to your left	
First and last name of student to your right	

Q1. [60 pts] Multiple Answer

Fill in the bubbles for **ALL correct choices**: there may be more than one correct choice, but there is always at least one correct choice. **NO partial credit**: the set of all correct answers must be checked.

(a) [4 pts] What is true of convolutional neural networks (CNNs)?

● A: Learned convolutional masks can act as edge detectors or line detectors.

● B: A convolutional layer of connections from a layer of m hidden units to a layer of m' hidden units has fewer weights than a fully-connected layer of connections from a layer of m hidden units to a layer of m' hidden units.

○ C: Let X be a 4×4 image that is fed through an average-pooling layer with 2×2 masks, producing a 2×2 output layer Y. Then for every unit/pixel X_{ij} and every unit/pixel $Y_{k\ell}$, the partial derivative $\partial Y_{k\ell}/\partial X_{ij}$ is equal to $1/4$.

● D: Some research on CNNs made a strong enough impact to win the Alan M. Turing Award.

A is mentioned in the lecture on CNNs.
B just says that convolutional layers aren't fully connected.
C is false. Although every non-zero entry of the matrix is equal to $\frac{1}{4}$, there are also many zeros within the $\frac{\partial Y}{\partial X}$ tensor stemming from the fact that some Y values do not depend at all on some other X values.
D is mentioned in the lecture on CNNs.

(b) [4 pts] What is true of regression algorithms?

○ A: All regression problems can be solved by solving a system of linear equations.

○ B: Least squares regression can be derived from maximum likelihood estimation if we assume that the sample points have a multivariate normal distribution.

○ C: Adding a feature with no predictive value to least squares regression is unwise because it increases the bias of the method.

● D: *Weighted* least squares regression can be derived from maximum likelihood estimation if we assume that some points' labels are noisier than others.

A: Logistic regression and Lasso are counterexamples. B: least squares regression assumes that the label noise is modeled by a Gaussian, not the sample points. C: no, it will not increase the bias, though it might decrease it. However, it is unwise, as it might increase the variance. D is true, weighted least squares regression can be motivated when different points have different noise variances.

(c) [4 pts] What is true of kernels?

○ A: Kernel algorithms require you to solve a linear system with a $D \times D$ matrix, where D is the length of the lifted sample points $\Phi(X_i)$.

○ B: Neural networks can be kernelized because there is always an optimal weight vector w that is a linear combination of the sample points.

○ C: Fitting and evaluating a kernel ridge regression model requires access only to the kernel matrix K and not the training points X_i nor $\Phi(X_i)$.

● D: The kernel perceptron algorithm with a Gaussian kernel returns a classifier similar to a smoothed version of k-nearest neighbor classification.

A: If that were true, kernel methods would be so slow nobody would use them.
B: Haha; the weight vector is generally not even the same length as the sample points.
C: To evaluate the decision/regression function on a test point z, we need access to the training points so we can compute kernels $k(X_i, z)$.
D: As stated on page 92 of Lecture 16.

(d) [4 pts] Which of the following statements are true of the Rayleigh quotient $Q(X, w) = w^\top X^\top X w/(w^\top w)$ for an arbitrary matrix $X \in \mathbb{R}^{n \times d}$ and vector $w \in \mathbb{R}^d$?

● A: $Q(X, w) \geq 0$ for all X and w.

● B: $Q(X, w) \leq \sigma^2_{\max}(X)$ for all X and w, where $\sigma_{\max}(X)$ is the greatest singular value of X.

● C: $Q(X, w)$ is maximized when w is the eigenvector of $X^\top X$ corresponding to the greatest eigenvalue.

○ D: $Q(X, w)$ is maximized when w is the eigenvector of $X^\top X$ corresponding to the smallest eigenvalue.

A is true because $w^\top X^\top X w = \|Xw\|^2$ cannot be negative, nor can $w^\top w = \|w\|^2$. C was stated in lecture. For the maximizing w, the length of Xw is $\sigma_{\min}$ times the length of w, hence $\|Xw\|^2$ is $\sigma^2_{\min}$ times $\|w\|^2$, so C implies B.

(e) [4 pts] Given $X \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^n$, consider solving the normal equations $X^\top X w = X^\top y$ for $w \in \mathbb{R}^d$. Let the SVD of X be $X = UDV^\top$. Let X^+ denote the Moore–Penrose pseudoinverse of a matrix X. Which of the following statements are certain to be true, for any values of X and y?

○ A: $w = (X^\top X)^{-1} X^\top y$ is a solution of the normal equations.

● B: $w = X^+ y$ is a solution of the normal equations.

● C: $w = VD^+ U^\top y$ is a solution of the normal equations.

○ D: Every solution w to the normal equations is a solution to $Xw = y$.

A is incorrect because $(X^\top X)$ might be singular, thus not invertible. But the pseudoinverse is always defined, and B is correct. C follows from B by writing out a formula for X^+. D is incorrect; $Xw = y$ does not always have a solution.

(f) [4 pts] Which of the following would be a reasonable cost function (by the criteria discussed in lecture) for choosing splits in a decision tree for two-class classification, where p is the fraction of points in Class C in a specified treenode?

- ● A: $-p \log_2 p - (1-p)\log_2(1-p)$
- ○ B: $p \log_2 p + (1-p)\log_2(1-p)$
- ○ C: $0.5 - |p - 0.5|$
- ● D: $p(1-p)$

A is the entropy (for two classes), which works well. B is minus entropy, which is convex and counterproductive; it favors bad splits. C is not strictly concave and is similar to the percent misclassified, and so is poor for the reasons given in class for the latter. D is the Gini impurity, also briefly mentioned in class, which works well.

(g) [4 pts] Which of the following are advantages to using AdaBoost with short trees (say, depth 4) over random forests with an equal number of tall trees (refined until the leaves are pure)?

- ○ A: AdaBoost is more robust against overfitting outliers in the training data.
- ● B: AdaBoost is faster to train.
- ○ C: AdaBoost is better at reducing variance than a random forest.
- ● D: AdaBoost is better at reducing bias than a random forest.

A: AdaBoost is worse at handling outliers because of the exponential loss function and the fact that it generally reaches 100% training accuracy. B is correct because AdaBoost uses shorter trees. C is wrong because bagging and randomizing the split choices are designed to reduce variance, whereas AdaBoost doesn't actively try to reduce variance and empirically sometimes overfits in practice. D is correct because AdaBoost is explicitly designed to reduce bias, and usually can reach 100% training accuracy given enough trees, whereas random forests tend to increase bias.

(h) [4 pts] What is true of k-nearest neighbor classifiers in a Euclidean space?

- ● A: A 1-NN classifier is more likely to overfit and less likely to underfit than a 10-NN classifier.
- ● B: In the special case of a two-dimensional feature space, it is possible to preprocess a data set so that nearest neighbor queries can be answered in $O(\log n)$ time.
- ● C: Sometimes finding k approximate nearest neighbors can be done much faster than finding the k exact nearest neighbors.
- ○ D: The k-d tree algorithm for nearest neighbor search can speed up 1-NN, but it doesn't work well for k-NN.

A: Of course. B: Yes, build a planar point location data structure on a Voronoi diagram. C: Of course. D: Wrong; k-NN is nearly as easy as 1-NN.

(i) [4 pts] What is true of bagging? (Note: not in a random forest; just bagging alone.)

- ● A: Bagging without replacement is more likely to overfit than bagging with replacement.
- ○ B: Bagging involves using different learning algorithms on different subsamples of the training set.
- ○ C: Bagging is often used with decision trees because it helps increase their training accuracy.
- ● D: Even with bagging, sometimes decision trees still end up looking very similar.

A: Bagging without replacement tends to make your subsamples more alike, which makes your trees more alike, rather than "decorrelating" them, so you're more likely to overfit. In the extreme case, if $n' = n$, all your trees will be identical. B: No, it's always the same learning algorithm. If you're using different learning algorithms, there's generally no reason to subsample. C: A single decision tree can achieve 100% training accuracy if we refine it enough and there are no two coincident training points from different classes, whereas bagging might not achieve 100% training accuracy. Bagging is used to reduce variance, not to reduce bias or increase training accuracy. D: True, if there are a few features that dominate/are very strong predictors. That's what motivated the invention of random forests.

(j) [4 pts] What is true of k-means clustering?

○ A: k-means is a supervised learning algorithm.

○ B: k-means clustering always converges to the same solution regardless of how clusters are initialized.

● C: Increasing k can never increase the optimal value of the k-means cost function.

● D: The k-medoids algorithm with the ℓ_1 distance is less sensitive to outliers than standard k-means with the Euclidean distance.

A is false; k-means clustering does not require any input labels. B is false; k-means is sensitive to initialization and converges to a local minimum which might not be the global minimum. C is true; adding more clusters can only decrease your loss for k-means (proven in Discussion Section 13). D is true. Penalizing ℓ_1 distances is generally less sensitive to outliers than penalizing ℓ_2 (Euclidean) distances.

(k) [4 pts] Which of the following statements about AdaBoost is true?

○ A: AdaBoost is a natural good fit with the 1-nearest neighbor classifier.

● B: AdaBoost can give you a rough estimate of the posterior probability that a test point is in the predicted class.

○ C: AdaBoost trains multiple weak learners that classify test points by equal majority vote.

● D: AdaBoost with an ensemble of soft-margin linear SVM classifiers allows the metalearner to learn nonlinear decision boundaries.

A: No. The 1-NN algorithm always gets 100% training accuracy (if there are no coinciding points from different classes), so what is there to boost? And how would you use the AdaBoost's weights in 1-NN?
B: True, from the continous output of the metalearner; see the lecture.
C: The weak learners are weighted based on their error rate.
D: True.

(l) [4 pts] You have n training points, each with d features. You try two different algorithms for training a decision tree.

1. The standard decision tree with single-feature (axis-aligned) splits, as covered in lecture.
2. A *randomized* decision tree—like a tree in a random forest, but there is only one tree and we don't use bagging. At each internal node of the tree, we randomly select m of the d features, and we choose the best split from among those m features.

Suppose you train both trees permitting no treenode to have depth greater than h. Upon completion, you find that in each tree, at least half its leaves are at depth h. What are the running times for training the standard decision tree and the randomized one, respectively? (Select exactly one option.)

○ A: $\Theta(dn2^h)$; $\Theta(mn2^h)$

○ B: $\Theta(dn2^h)$; $\Theta(dn2^h)$

● C: $\Theta(dnh)$; $\Theta(mnh)$

○ D: $\Theta(dnh)$; $\Theta(dnh)$

A training points X_i participates only in the treenodes in the path from the root to the leaf node that contains X_i, and there are at most $h + 1$ such treenodes. If we sum over all n training points, there are $\Theta(nh)$ point-treenode pairs (consisting of a point that participates in a treenode).

During training (tree construction), the cost of processing a treenode in the standard tree is d times the number of training points that participate in the treenode. Summing this over treenodes, we have $\Theta(nh)$ point-treenode pairs to process each requiring d time, so the overall running time is in $\Theta(dnh)$.

The cost of processing a treenode in the randomized tree is m times the number of training points that participate, so the overall running time is in $\Theta(mnh)$.

(m) [4 pts] Suppose you have a fixed dataset and want to train a decision tree for two-class classification. If you increase the maximum depth of the decision tree, which of the following are possible effects?

● A: The test accuracy goes up.

○ B: The training accuracy goes down.

○ C: The number of pure leaves is reduced.

● D: The time to classify a test point increases.

A: this is possible, and is generally the goal of making a decision tree more complex.
B: accuracy on the training set cannot go down.
C: pure leaves in the short tree remain pure leaves in the large tree.
D: adding layers adds more nodes for the training algorithm to traverse.

(n) [4 pts] What is true of the shatter function $\Pi_H(n)$, where n is the number of training points?

● A: It either grows polynomially with n or is 2^n for all n, but it can never be in between those extremes.

● B: We use it to compute an estimate of how many training points we need to ensure we choose a hypothesis with nearly optimal risk with high probability.

● C: For a linear classifier in $\mathbb{R}^d$, the shatter function grows polynomially with n.

○ D: For a power set classifier, the shatter function grows polynomially with n.

All of these come from the lecture.

(o) [4 pts] What is true of human neurology?

○ A: The output of a unit in an artificial neural network is roughly analogous to the voltage of an axon at the synapse.

● B: A connection in an artificial neural network is roughly analogous to a synapse in the brain.

○ C: The brain is made of general-purpose neurons, each of which could be trained to do any job that neurons do.

● D: The visual cortex has neurons primarily devoted to detecting lines or edges.

A is wrong—the output is analogous to the firing rate, not the voltage. B and D are empirically true. C is very wrong. The brain has many functional regions that are specialized to do very different things. A neuron in the visual cortex can't be trained to do emotion.

Q2. [20 pts] Decision Tree Construction

You want to figure out what makes people click on online advertisements, so you try a variety of different advertisements, and record user behavior in the following table, which is our training set.

Advertisement	Animated?	Popup?	Colorful?	Clicked
1	Yes	Yes	No	No
2	No	Yes	Yes	No
3	No	Yes	No	No
4	No	No	Yes	Yes
5	Yes	No	Yes	Yes
6	Yes	No	No	Yes
7	No	No	Yes	Yes
8	No	No	No	No

The middle three columns ("Animated," "Popup," "Colorful") are the features of each advertisement. The last column, "Clicked," is the label, specifying whether or not a user clicked on the advertisement. You want to predict the "Clicked" label for other advertisements not in this training set.

(a) [8 pts] **Which feature** should you split on at the root of the decision tree to maximize the information gain? **Write an expression for the information gain** of the best split. (Your expression can contain logarithms and fractions.)

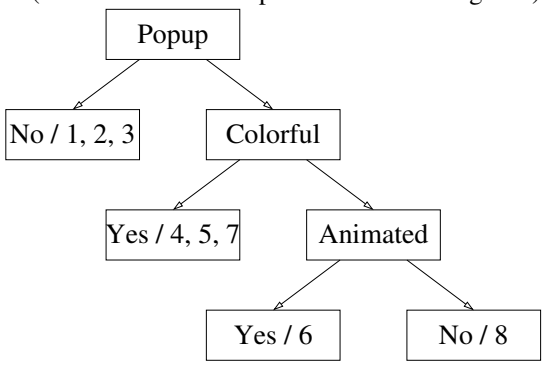
Split on "Popup".

[Explanation, not required for points: It gives you a pure leaf of 3 and a leaf of 5 with a 4-1 split. "Animated" gives you a leaf of 3 with a 2-1 split and a leaf of 5 with a 3-2 split; clearly higher entropy in both children. "Colorful" gives you two leaves of 4 with 3-1 splits. As a 3-1 split has higher entropy than a 4-1 split, and the higher entropy is weighted by all 8 training points (rather than just 5 points for "Popup"), "Colorful" has higher weighted entropy than "Popup."]

The information gain is

$$-\frac{4}{8}\log_2\frac{4}{8}-\frac{4}{8}\log_2\frac{4}{8}+\frac{3}{8}(0)+\frac{5}{8}\left(\frac{4}{5}\log_2\frac{4}{5}+\frac{1}{5}\log_2\frac{1}{5}\right).$$

(b) [8 pts] **Draw the decision tree** that maximizes information gain at each split. Your drawing should include the leaves and the training points they store. (Do not include entropies or information gains.) Stop splitting at pure nodes.



(c) [4 pts] Suppose you are using AdaBoost with stumps, which are trees with just one split. In the first iteration, AdaBoost assigns each advertisement a weight of $1/8$, and it constructs a one-split tree with the split you specified in part (a). **What weights does AdaBoost assign** to the eight advertisements for the **second iteration**? Show your work.

The first tree misclassifies only Advertisement 8, so the weighted error rate of the first tree is $1/8$. The weight of Advertisement 8 is increased by a factor of $\sqrt{(1-\text{err})/\text{err}} = \sqrt{7}$, so its new weight is $\sqrt{7}/8$. The weights of Advertisements 1–7 are decreased by a factor of $\sqrt{7}$, so their new weight is $1/(8\sqrt{7})$.

Q3. [12 pts] Review of k-means Clustering

Let's see how well you remember k-means clustering, also known as Lloyd's Algorithm. As usual, the input is a set of n sample points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and an integer k. There are no input labels; we want to assign each sample point X_j a label $y_j \in \{1, 2, \ldots, k\}$, which can be interpreted as "X_j is assigned to cluster y_j." The means of the clusters are written $\mu_1, \mu_2, \ldots, \mu_k$.

(a) [4 pts] **What is the cost function** that k-means clustering tries to minimize? Write it in terms of the X_j's, the y_j's, and the μ_i's. (Not the formula for the within-cluster variation, please; a formula that uses the means. We are flexible about what notation you use to sum the terms in the cost function; please explain it if it's not obvious.)

One way to write it is

$$\sum_{i=1}^{k} \sum_{y_j=i} \left\| X_j - \mu_i \right\|^2 .$$

Another fine way is

$$\sum_{j=1}^{n} \|X_j - \mu_{y_j}\|^2.$$

(b) [5 pts] Consider the step of the algorithm where the labels y_j are held fixed while the cluster means μ_i are updated. I asserted in lecture that it is easy to show with calculus that if we want to minimize the cost function, we should **choose each μ_i to be the mean (centroid) of the sample points assigned to cluster i.** Please do that calculus and **show that this claim is correct**. (Make sure you explain your notation for counting the points in a cluster.) **Show your work and don't skip any steps of the derivation.**

Let n_j be the number of sample points assigned to cluster j. If we use the first version of the cost function, we have

$$\frac{\partial J}{\partial \mu_i} = \sum_{y_j=i} (\mu_i - X_j).$$

Setting that derivative to zero gives us

$$\begin{aligned} \sum_{y_j=i} \mu_i &= \sum_{y_j=i} X_j; \\ n_j \mu_i &= \sum_{y_j=i} X_j; \\ \mu_i &= \frac{1}{n_j} \sum_{y_j=i} X_j, \end{aligned}$$

which indeed is the mean of the sample points assigned to cluster i.

If we use the second version of the cost function, we have

$$\frac{\partial J}{\partial \mu_i} = \frac{\partial}{\partial \mu_i} \sum_{y_j=i} \|X_j - \mu_{y_j}\|^2 = \sum_{y_j=i} (\mu_i - X_j),$$

and the rest of the derivation proceeds in the same way.

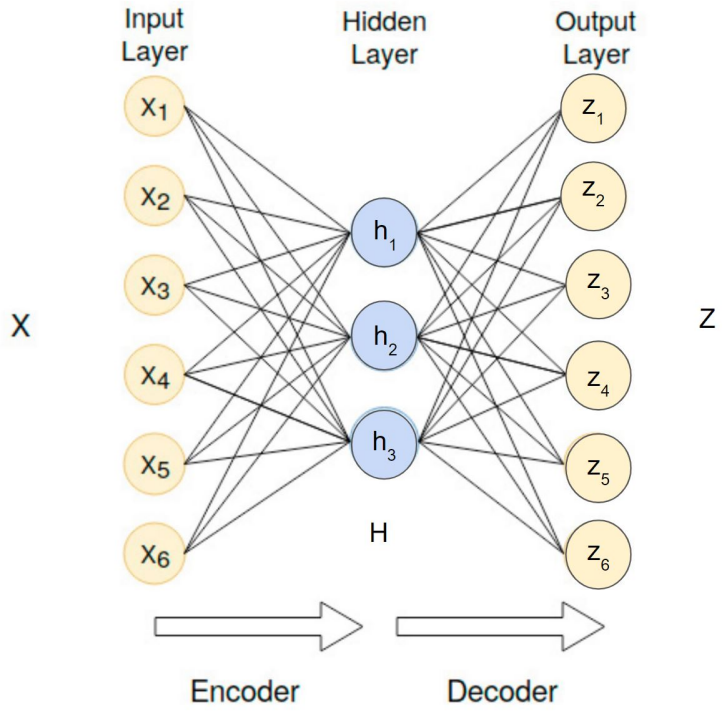
(c) [3 pts] Consider the step of the algorithm where the cluster means μ_i are held fixed while the labels y_j are updated. Sometimes a sample point X_j has several cluster means that are equally close. In class I said, "If there's a tie, and one of the choices is for X_j to stay in the same cluster as the previous iteration, always take that choice." **What could conceivably go wrong** if you don't follow that advice?

If a sample point keeps switching back and forth between two (or more) cluster means that are equally close, the k-means algorithm might never terminate even though it has can make no further progress in reducing the cost function.

Q4. [20 pts] Backpropagation through an Autoencoder

An autoencoder is a type of neural network used to learn efficient encodings of data. We present a vector $x \in \mathbb{R}^d$ at the network's input units. A fully connected layer of edges produces a lower-dimensional vector of hidden units $h \in \mathbb{R}^m$, where $m < d$. Another fully connected layer produces an output vector $z \in \mathbb{R}^d$. Usually, the output label associated with x is x—in other words, the goal of training is simply to reproduce the input vector at the output. What makes this difficult is that there are so few hidden units. The network has to learn how to compress the information in the training set so it can be represented in a lower-dimensional space without losing much information. This is not always possible; it depends on the data. If successful, the hidden units are considered to be a *latent representation* of the input data.

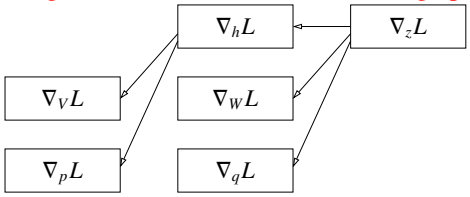
We will use sigmoid/logistic units and a notation similar to that used in lecture, but we will break out the bias terms into separate vectors. Specifically, $h = s(Vx + p)$ where $V \in \mathbb{R}^{m \times d}$ is a matrix of weights connecting x to h, $p \in \mathbb{R}^m$ is a vector of bias terms, and $s(\cdot)$ denotes the logistic (sigmoid) function $s(\gamma) = 1/(1 + e^{-\gamma})$ applied element-wise to a vector. Similary, $z = s(Wh + q)$ where $W \in \mathbb{R}^{d \times m}$ and $q \in \mathbb{R}^d$. As the goal of training is to make the output vector z match the input vector x very tightly, we use the loss function $L(z, x) = \|z - x\|_2^4$.



(a) [6 pts] See the list of gradients below. **Draw a directed graph** whose vertices are the gradients (from this list) that are calculated during backpropagation to train our autoencoder. Do not include gradients that are not computed as part of backpropagation. If a gradient G is used by backpropagation to speed up the calculation of another gradient G', draw an arrow from G pointing to G'. (In other words, $G \rightarrow G'$ means G must be computed before G' if you want all the speed advantages of backpropagation. The graph will show us a partial ordering of the gradient computations.)

$$\nabla_x L, \nabla_h L, \nabla_z L, \nabla_V L, \nabla_p L, \nabla_W L, \nabla_q L$$

The graph should omit $\nabla_x L$, but the other six gradients should be vertices of the graph.



(b) [4 pts] **Derive an expression for** $\nabla_z L$ in terms of z and x (or components of z and x).

If you remember that $\nabla_w \|w\|^4 = 4\|w\|^2 w$, then the simplest derivation is to set $w = z - x$. Then $L(z, x) = \|w\|^4$ and by the chain rule, $\frac{\partial L}{\partial z_i} = \nabla_w L \cdot \frac{\partial}{\partial z_i} w = 4\|w\|^2 w_i$, so $\nabla_z L = 4\|w\|^2 w = 4\|z - x\|^2 (z - x)$.

A direct way to derive the gradient is to let $v = \|z - x\|^2$. Then $L(z, x) = \|z - x\|^4 = v^2$ and by the chain rule, $\nabla_z L = \frac{\partial L}{\partial v} \nabla_z v = 2v \cdot 2(z - x) = 4\|z - x\|^2 (z - x)$.

Another way to derive this is to write $L(z, x) = \left(\sum_{i=1}^{d} (z_i - x_i)^2 \right)^2$.

Then by the chain rule, $\frac{\partial L}{\partial z_i} = 2 \left(\sum_{i=1}^{d} (z_i - x_i)^2 \right) \cdot 2(z_i - x_i) = 4\|z - x\|^2 (z_i - x_i)$, so $\nabla_z L = 4\|z - x\|^2 (z - x)$.

(c) [5 pts] **Derive an expression for** $\frac{\partial z_j}{\partial h_i}$ (the jth unit of z with respect to the ith unit of h). Express your final answer in terms of W and z (or components of W and z; if you need a notation for a column or row of W, explain your notation). **Show every step in the derivation** (i.e., include the steps that were skipped in lecture, except the derivation that $s' = s(1 - s)$, which you may take for granted).

Recall that $z = s(Wh + q)$ and that the derivative of the sigmoid function is $s'(x) = s(x)(1 - s(x))$. Hence $z_j = s(W_j \cdot h + q_j)$ and by the chain rule,

$$\frac{\partial z_j}{\partial h_i} = s'(W_j \cdot h + q_j) \frac{\partial}{\partial h_i} (W_j \cdot h + q_j) = z_j (1 - z_j) W_{ij},$$

where W_j is row j of W.

(d) [5 pts] **Derive an expression for** $\frac{\partial L}{\partial x_k}$**.** Express your final answer in terms of V, h, and $\nabla_h L$ (or components of V, h, and $\nabla_h L$). **Show every step in the derivation** (i.e., include the steps that were skipped in lecture, except $s' = s(1 - s)$).

By the chain rule,

$$\frac{\partial L}{\partial x_k} = \frac{\mathrm{d}L}{\mathrm{d}x_k} + \sum_{i=1}^{m} \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial x_k},$$

which could alternatively be written $\frac{\mathrm{d}L}{\mathrm{d}x_k} + (\nabla_h L) \cdot \frac{\partial h}{\partial x_k}$. Recall that $h = s(Vx + p)$, hence $h_i = s(V_i \cdot x + p_i)$ and by the chain rule,

$$\frac{\partial h_i}{\partial x_k} = s'(V_i \cdot x + p_i) \frac{\partial}{\partial x_k} (V_i \cdot x + p_i) = h_i (1 - h_i) V_{ik},$$

Therefore,

$$\frac{\partial L}{\partial x_k} = 4\|x - z\|^2 (x_k - z_k) + \sum_{i=1}^{m} \frac{\partial L}{\partial h_i} h_i (1 - h_i) V_{ik}.$$

Q5. [18 pts] Musings

(a) [4 pts] It's well known that a memory in the brain may be "distributed" rather than "localized." What this means is that there is no one small portion of the brain where the memory is located. **What evidence do we have of that?**

Experiments in animals (especially rats) that introduce lesions in various parts of their brain never succeed in causing an animal to forget behaviors like running a maze (unless the lesion is so large it's profoundly disabling).

(b) [4 pts] Memories in neural networks can also be "distributed." **What technique** we have learned in lecture that actively forces a neural net to distribute its memories? **How does it achieve this?**

Dropout.

During training, every unit will be turned off for a while, and the other units will need to learn to produce the correct answer without it. If training is successful, the output won't rely on any single unit.

(We'll give part marks for "convolutional neural networks" if the shared weights are mentioned or alluded to, but that's not really a good answer: the weights may be shared, but they're only stored in one place.)

(Any answer of the form "you can't do dropout if you have no hidden units" is missing the point completely and gets no credit.)

(c) [4 pts] However, not every memory in every neural network is distributed. If a neural network has only two layers of units—input and output, with no hidden units—then the memories it stores are entirely local, not distributed. **Explain why.**

There is only one path from any single input to any single output.

(d) [6 pts] Recall that in the lecture on principal components analysis, we saw a projection of the genetics (single nucleotide polymorphisms, abbreviated to SNPs, to be specific) of various old-stock Europeans onto a two-dimensional subspace, and we saw that it bore some resemblance to the geography of Europe. A skeptic might claim that distinct European ethnicities (e.g., French, Norwegian, Italian) do not exist, as "There is no single SNP that can identify the ethnicity of a European." The latter half of that sentence is true; probably there are no 10 SNPs that can do it.

Explain how we, as machine learning researchers, might generate evidence in favor of the existence of distinct ethnicities, despite the lack of a single SNP that distinguishes them, and despite the fact that some people cannot be classified unambiguously (because of multiple ancestry). **Also, explain** what insights random projections give us about the dimensions of the spaces we can use to answer such questions.

It is possible—easy, even—to have well-separated, tight clusters of points in a high-dimensional space, yet when they are projected to a low dimensional subspace, they appear to have a lot of overlap. You need to see them in the high-dimensional space to see that they are well separated. For a good example of what happens in low-dimensional projections, recall from lecture the PCA projection of the MNIST digits to 2D. It's easy for us to distinguish these digits by eye, so we know that the high-dimensional representation distinguishes them from each other cleanly. Yet in a 2D PCA projection, there is not nearly enough information/structure left to tell them apart.

Clustering algorithms can identify potential groupings; to "generate evidence in favor of the existence of distinct ethnicities," we could, for instance, quantify how close the average cluster member is to its own cluster center versus other cluster centers.

What we learn from random projections is that if we project the SNPs down to a space of dimension 10,000, the distances between points are well enough preserved that we will probably see the clusters and still be able to distinguish European ethnicities; but if we project them down to a space of dimension 10 (or in the extreme, 1), we will probably not. Hence, the argument that "There are no 10 SNPs that can identify the ethnicity of a European" does nothing to refute the idea that in a high-dimensional space, the clusters are robust.

Q6. [20 pts] Boosting with a Squared Loss Function

Recall that AdaBoost trains T classifiers $G_1, ..., G_T$ iteratively with the training points re-weighted in each iteration. Given a test point z, the final metalearner's output is a linear combination of these learners,

$$M_T(z) = \sum_{t=1}^{T} \beta_t G_t(z).$$

AdaBoost trains the metalearner with an exponential loss function. In this problem, we explore another common boosting algorithm known as *Gradient Boosting*. Let $\rho = M_T(z)$ be a prediction made by the metalearner, and let $\ell \in \{+1, -1\}$ be the real-world label for the same point z. Gradient Boosting replaces the exponential loss function with the squared error loss,

$$L(\rho, \ell) = (\rho - \ell)^2.$$

We will do classification with two classes. As usual, assume we are given n training points $X_1, X_2, \ldots, X_n \in \mathbb{R}^d$ and a vector $y \in \mathbb{R}^n$ of labels, with $y_i = \pm 1$.

(a) [6 pts] **Write down the total empirical risk** of the Gradient Boosting metalearner with a squared error loss function. Express your answer in terms of the β_t's, G_t's, and y_i's.

The risk is the average loss, $\frac{1}{n}\sum_{i=1}^{n} L(M_T(X_i), y_i)$. Plugging in the squared loss, we get

$$\text{risk} = \frac{1}{n}\sum_{i=1}^{n}\left(\sum_{t=1}^{T}\beta_t G_t(X_i) - y_i\right)^2.$$

(b) [8 pts] Suppose Gradient Boosting is in iteration T, and it has just trained a weak learner G_T but it has not yet determined the optimal coefficient β_T. (We assume that $\beta_1, \ldots, \beta_{T-1}$ are already fixed constants that can't be changed during iteration T.) **Derive an expression for β_T that minimizes the metalearner's empirical risk.** Show your work and simplify your answer as much as possible. *Hint:* Rewrite the risk in terms of M_{T-1}, the metalearner from iteration $T-1$. (That way you don't have to write a summation repeatedly. Note that there's more space to write on the next page.)

Find the β_T that minimizes the risk by deriving $d\,\text{risk}/d\beta_T$ and setting it to zero.

$$\begin{aligned}
\text{risk} &= \frac{1}{n}\sum_{i=1}^{n}\left(\beta_T G_T(X_i) + \sum_{t=1}^{T-1}\beta_t G_t(X_i) - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\beta_T G_T(X_i) + M_{T-1}(X_i) - y_i\right)^2 \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(\beta_T^2 G_T^2(X_i) + 2\beta_T G_T(X_i) M_{T-1}(X_i) - 2y_i\beta_T G_T(X_i) + M_{T-1}^2(X_i) - 2y_i M_{T-1}(X_i) + y_i^2\right). \\
\frac{d\,\text{risk}}{d\beta_T} &= \frac{1}{n}\sum_{i=1}^{n}\left(2\beta_T G_T^2(X_i) + 2G_T(X_i)M_{T-1}(X_i) - 2y_i G_T(X_i)\right).
\end{aligned}$$

Setting $d\,\text{risk}/d\beta_T = 0$ gives

$$\begin{aligned}
\beta_T \sum_{i=1}^{n} G_T^2(X_i) &= \sum_{i=1}^{n}\left(y_i G_T(X_i) - G_T(X_i)M_{T-1}(X_i)\right); \\
\beta_T &= \frac{\sum_{i=1}^{n} G_T(X_i)(y_i - M_{T-1}(X_i))}{\sum_{i=1}^{n} G_T^2(X_i)}.
\end{aligned}$$

(You may continue part (b) here.)

(c) [6 pts] In iteration T, before we compute β_T, Gradient Boosting has to train a weak learner G_T. Recall that the metalearner output $M_T(X_i)$ is a continously-valued real number, but the output of a weak learner $G_T(X_i)$ can only be $+1$ or -1. A paradoxical outcome of the squared error loss function is that if you truly want to minimize the empirical risk, the metalearner should provide the wrong label for some of the training points to the weak learner G_T during training! **Which training points should be given the wrong label when G_T is trained?** (Note: we'll accept an answer written in plain English, but we'll also accept an answer written in the form of a mathematical set. Your choice.)

If training point i has $y_i = +1$ and $M_{T-1}(X_i) > 1$, then $M_{T-1}(X_i)$ is too large and making $M_T(X_i)$ even greater would make the risk larger. So training G_T with a label of -1 on point X_i will help bring $M_T(X_i)$ closer to $+1$.

Symmetrically, if training point i has $y_i = -1$ and $M_{T-1}(X_i) < -1$, training G_T with a label of $+1$ on point X_i will help.

One short way to write the set of indices of the points that should be given the wrong label is

$$\{i : y_i M_{T-1}(X_i) > 1\}.$$

This by itself will earn full points, but it isn't necessary for full points; a plain English explanation will do.